
White Paper

# High Octane CRC Generation with the Intel Slicing-by-8 Algorithm

# Introduction

Introduction 2

CRC's and Their Importance in Storage Area Networks 3

iSCSI and the Need for Speed 4

Tripling CRC Performance with the Slicing-by-8 Algorithm 5

The CRC Performance Benefits of the Slicing-by-8 Algorithm 8

Conclusion 11

Cyclic Redundancy Codes (CRCs) perform mathematical calculations on blocks of data to check for errors when files are transferred from one location to another. Networking protocols use CRCs to verify data received is the same as data sent during the production, transmission, processing and storage of digital data. CRCs are popular because they are simple to implement, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

CRCs are receiving greater attention today because of the rapid emergence of the Internet Small Computer System Interface (iSCSI). The iSCSI protocol uses TCP/IP and the ubiquitous Ethernet interface. It can support multi-gigabit storage traffic throughput, enabling low-cost centralization of storage without the expense and incompatibility issues associated with setting up expensive Fibre Channel storage area networks. Consequently, iSCSI is fast becoming a vital ingredient for IP-based block network storage.

CRCs play an important role in error detection in iSCSI implementations. Unfortunately, they are also a major bottleneck in iSCSI processing performance. CRC computation adds expensive overhead to every transmission. This paper discusses the importance of iSCSI, CRCs, and a new "Slicing-by-8" algorithm Intel has developed that triples the performance of existing software-based CRC implementations while using just an 8 K cache footprint. We include instructions for software developers interested in getting the Slicing-by-8 algorithm distributed free by Intel.

# CRCs and Their Importance in Storage Area Networks

While most people believe networks are error-free, they do have errors. Hence the need for CRCs. Networks can experience noise and packet loss. It's simply a fact of life in packet transmission. CRCs significantly reduce the impact of errors by detecting them efficiently. Once errors are detected, the iSCSI layer triggers retransmission for quick recovery.

CRCs are essentially "industrial strength" checksum algorithms providing an extremely high level of data integrity assurance. CRCs are applied to do everything from minor tasks, such as making sure data accurately copies to a rewritable CD, to critical tasks, such as ensuring a company's complete financial records for a year are accurately backed up on a server. In most applications, this makes CRCs worth the price of their additional computational complexity.

CRCs are receiving much more attention now that iSCSI solutions are becoming popular for mission-critical applications such as storage area networks (SANs). These high-speed subnetworks of shared storage devices make a group of storage devices available to all servers on a LAN or WAN. As more storage devices are added to a SAN, they too become accessible from any server in the larger network. Using SANs instead of servers with direct attached storage enables IT departments to provide a pathway between the users and stored data while dedicating a much greater proportion of server performance to the business applications housed on them.

Obviously, with mission-critical or even just daily business information stored on a SAN, it's vital to know that files are valid and reliable. In an iSCSI network, that's done by having a target (i.e., a storage server) calculate a CRC as data blocks are sent and then append this CRC to the blocks for transfer across the network. The receiver (called the initiator) calculates a CRC on its own (using the same CRC algorithm) and compares it to the CRC calculation appended on the data blocks received. If they match, data integrity is assured. The opposite is done when an initiator writes a data block to the target. The initiator creates the CRC and appends it to the data. The target, or storage server, calculates a CRC on the incoming data block and compares. If they don't match, an error is flagged and the data blocks are discarded. The iSCSI protocol may then initiate retransmission of the original blocks.

# iSCSI and the Need for Speed

The most prevalent technology for SAN installations has been the Fibre Channel (FC) Protocol. But this technology has limited the implementation of SANs. It isn't affordable for many organizations and its deployment is complex. These disadvantages have spurred the enthusiasm for the iSCSI protocol.

One of the most compelling selling points of iSCSI solutions is that they enable inexpensive storage networking. Because the iSCSI protocol specifies how to run SCSI commands over existing TCP/IP infrastructure, it provides a lower cost alternative than installing a fibre channel network. This makes iSCSI solutions a superior choice in many cases for SANs. (As for the efficacy of using TCP/IP infrastructure, it's worth noting that some of the largest banks in the U.S. run their entire banking operations via IP networks.)

The advantages of the iSCSI protocol include:

- The enormous IP infrastructure already in place.
- Ability to deploy less expensive, easier-to-use devices that are already part of the large installed Ethernet ecosystem (silicon, boards, switches, management software, and trained technicians). This translates into easier management and interoperability.
- Straightforward setup of SANs and easier remote replication of storage for branch offices. IT administrators, already familiar with deployment and provisioning of Ethernet and TCP/IP networks, can easily set up a SAN at their headquarters or remote branch.
- iSCSI software drivers enable practically any system with a NIC to be added to the network.

One hurdle limiting broader acceptance of iSCSI technology is that systems have to be able to deal with large amounts of network input/output (I/O). Because the iSCSI protocol is essentially a wrapper around standard SCSI commands, it places an additional burden on a server's microprocessor. These commands must be received in order, and each command or data packet must be wrapped and unwrapped. The TCP transport provides reassembled data in the correct order and provides a checksum for the data in each packet. For extra data protection against rare events—such as a router fragmenting a packet and generating a new valid checksum for a fragmented packet having a changed data bit—it's recommended that the CRC option be activated. This adds significantly more overhead to the process.

Performance is a major concern for anyone, particularly enterprise customers. Today's IT departments spend large amounts of money on a variety of different projects. To get maximum value requires maximum utilization. One important way to achieve such utilization is through performance enhancements that enable you to do more with less hardware and minimize capital investment. In this context, the extra overhead of CRCs deserves special scrutiny. And even better, a solution.

# Tripling CRC Performance with the Slicing-by-8 Algorithm

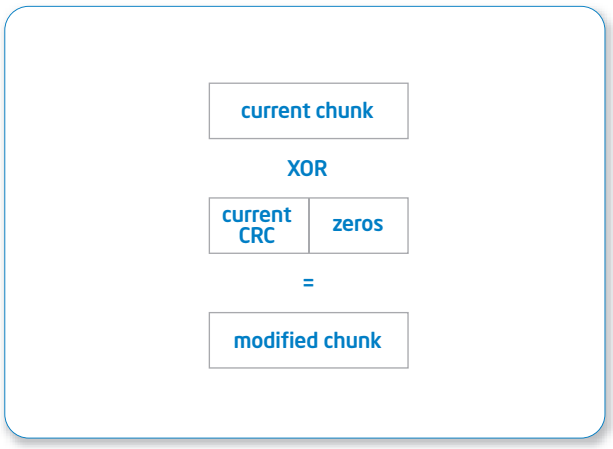
In examining any performance-enhancing solution, it's first important to note that CRCs today are commonly generated in software. This is for many reasons. Perhaps the most important is that software-only iSCSI implementations enable the use of general purpose processors without specialized iSCSI circuitry. Software-only implementations also scale better with the CPU clock speed and the number of processing units available. This last advantage—scaling better with available processing units—will grow in importance as more multi-core platforms become available. Another reason software-only iSCSI implementations are popular is the limited number of specialized CRC generation circuits on many commercial host, network and server chipsets.

In the past, a number of software-based algorithms have been proposed and tried to accelerate the CRC generation process. Among these algorithms, the "state of the art" and most commonly used today is the Sarwate algorithm. Its primary shortcoming is that it only reads 8 bits at a time from a stream. This is because the Sarwate algorithm was designed at a time when most computer architectures only allowed arithmetic operations between 8-bit quantities. That restriction no longer holds true. Computer architecture technology has progressed to the point where arithmetic operations can be performed efficiently between 32- or 64-bit quantities. Consequently, it's time to make use of this advantage.

As mentioned above, the CRC generation process is one of the most time-consuming parts of iSCSI processing. CRC generation is costly because it requires several logical operations to be performed on a byte-by-byte basis for each block of data. To understand how our new CRC Slicing-by-8 generation algorithm addresses this bottleneck, we need to explain how CRC algorithms work. CRC algorithms treat each bit stream as a binary polynomial and describe it by the remainder of a calculation dividing this binary polynomial with a standard 'generator' polynomial. The binary words (a "binary word" is a sequence of binary values) corresponding to this remainder are then transmitted together with the bit stream associated with the binary polynomial. The length of the remainder in bits is equal to the length of the generator polynomial minus one. At the receiver side, CRC algorithms verify that the remainder is the correct remainder by repeating the calculation.

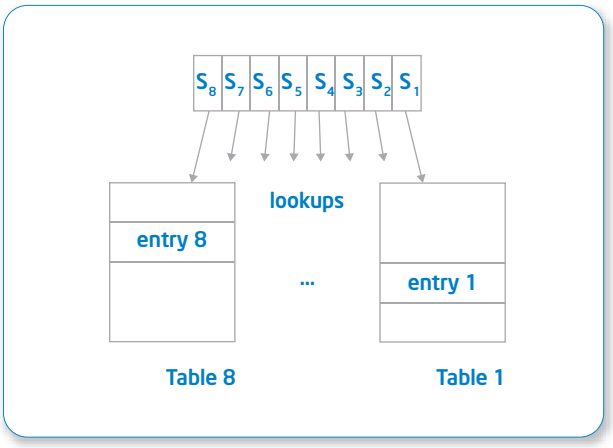
The long division process is a compute-intensive operation because it requires, in the worst case, one shift operation and one XOR logical operation (an operation that can be executed on two or more binary strings) for every bit of a bit stream. Most software-based CRC generation algorithms, however, perform the long division quicker than the bit-by-bit marking process described above. One commonly used technique for accelerating the long division process is to pre-compute the current remainder that results from a group of bits and place the result in a table. Before the beginning of the long division process, all possible remainders which result from groups of bits are pre-computed and placed into a lookup table. In this way, several long division steps can be replaced by a single table lookup step. Additional table lookups can further replace subsequent long division steps. To avoid using large tables, table-driven CRC acceleration algorithms typically read no more than 8 bits at a time.

**Figure 1:** The Slicing-by-8 Algorithm
For every 64-bit chunk of an input stream the algorithm performs four steps:

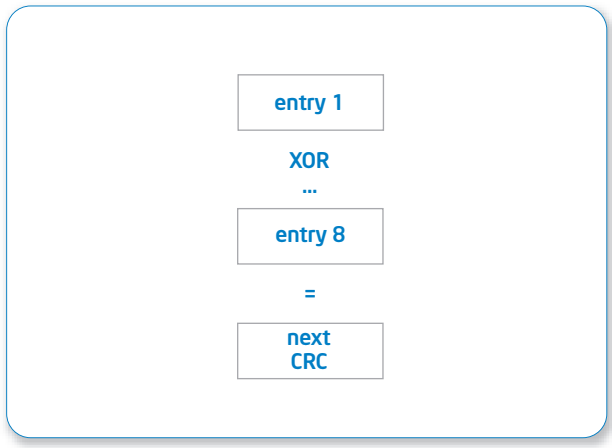



**Step 1:** An XOR operation is performed between the 32 most significant bits of the chunk and the current CRC value.


**Step 2:** The 64-bit value produced from this XOR operation is then sliced into 8 slices of equal length.




**Step 3:** Each slice is used for accessing a separate lookup table. The lookup tables used by the algorithm store the remainders from the division of all possible 8-bit numbers shifted by a variable number of bits to the left with the generator polynomial. The offset values used for calculating the table entries begin with 32 for Table 1 and increase by 8 for every table.




**Step 4:** The values returned from all table lookups are XOR-ed to one another producing the CRC value used in the next iteration of the algorithm's main loop.

The main disadvantage of existing table-driven CRC generation algorithms is this memory space requirement when reading a large number of bits at a time. For example, if we want to achieve acceleration by going beyond reading 8 bits at a time to 32 bits at a time, table-driven algorithms require storing pre-computed remainders in a table of 232. That's over 4 G in entries! To solve this problem, Intel has devised a new algorithm that slices the CRC value produced in every iteration, as well as the data bits read, into small terms. These terms are used as indexes for performing lookups on different tables in parallel. In this way, the Slicing-by-8 algorithm is capable of reading 64 bits at a time, as opposed to 8, while keeping its memory space requirement to 8 KB. In other words, bit slicing is Intel's solution to the memory explosion problem associated with existing table-driven algorithms.

The benefit from slicing comes from the relatively large cache unit size of today's processor architectures. These cache units are capable of storing moderate-size tables, such as the 8 KB tables required by the Slicing-by-8 algorithm, though not sufficient for storing tables associated with taking significantly larger strides, such as the 16 GB tables that would be associated with 32-bit strides. Why not store the tables in an external memory unit? The latency is far too high. A DRAM memory access, for example, requires several hundreds of clock cycles for an Intel® Pentium® M processor to complete, whereas completing an access to a first-level cache memory unit requires less than five clock cycles. For this reason, the processing cost associated with slicing is typically insignificant when compared to the cost of accessing off-chip memory units.

Slicing is also important because it reduces the number of operations performed for each byte of an input stream when compared to the Sarwate algorithm. Fewer operations make the Sicing-by-8 algorithm much faster than the Sarwate algorithm. For each byte of an input stream, the Sarwate algorithm performs the following:

1. An XOR operation between a byte read and the most significant byte of the current CRC value
2. A table lookup
3. A shift operation on the current CRC value
4. An XOR operation between the shifted CRC value and the word read from the table

Compare this to the Slicing-by-8 algorithm. For every byte of an input stream the Slicing-by-8 algorithm only has to perform:

1. A table lookup
2. An XOR operation

# The CRC Performance Benefits of the Slicing-by-8 Algorithm

To evaluate the performance benefits, we added the Slicing-by-8 CRC implementation to a test iSCSI stack. Figure 2 compares the performance of different-sized iSCSI read runs with two different CRC generation algorithms. The read I/O workload size ranges from 512 B to 8 KB. The horizontal axis represents the read I/O size, while the vertical axis shows the total cycles spent on computing a CRC over that I/O size.

As seen in the graph, the Slicing-by-8 algorithm requires fewer cycles than the Sarwate algorithm to compute the CRC at all data points. Specifically, for the 8KB data point, the Slicing-by-8 algorithm takes about 2.15 cycles per byte to generate a CRC value, while the Sarwate algorithm takes 6.55 cycles per byte. Thus, Slicing-by-8 accelerates the CRC generation process by a factor of three, as compared to Sarwate. The Sarwate algorithm requires executing 35 IA-32 instructions in order to validate 32 bits of data, whereas the Slicing-by-8 algorithm requires 13 instructions only. This is the reason why the Slicing-by-8 algorithm is three times faster than the Sarwate algorithm.

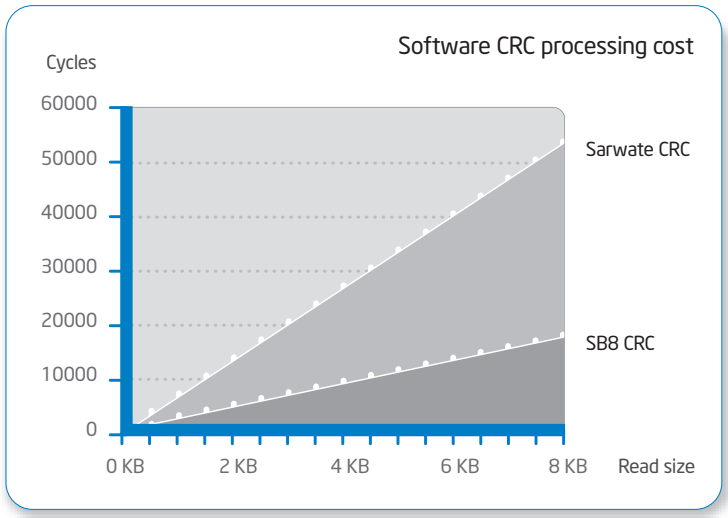



**Figure 2:** SB8 CRC versus Sarwate CRC
Data: Intel Corp.

Figure 3 compares the iSCSI processing cost for a single data-in (read) protocol data unit (PDU) at the initiator (client) between the Sarwate CRC and the Slicing-by-8 CRC generation algorithm. The horizontal axis represents the size of the read I/O command issued by the initiator. The primary vertical axis (i.e., the axis on the left) represents the relative processing costs of the protocol, the CRC generation process, and the data copies. The secondary vertical axis (i.e., the axis on the right) represents the absolute number of cycles spent on iSCSI processing for a single data-in PDU. Since the read I/O workload size varies from 512 B to 8 KB, and the PDU size is set to 8 KB, this implies that each iSCSI read command results in the reception of a single data-in PDU.

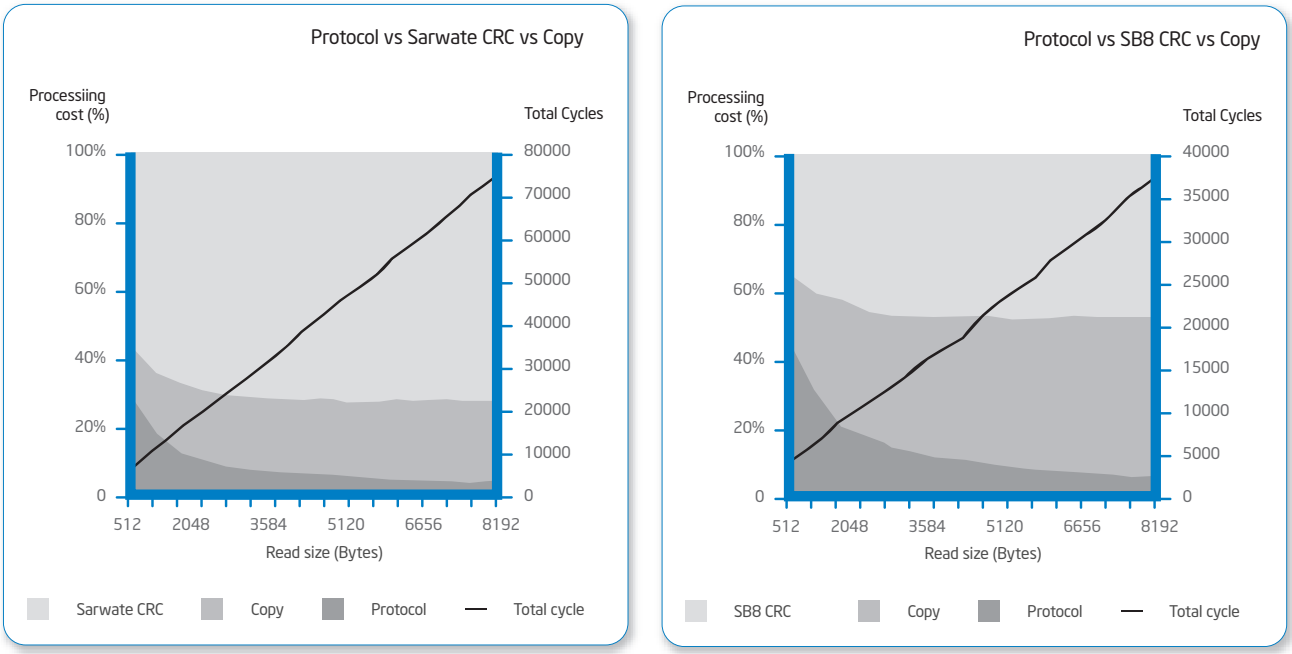



**Figure 3:** Comparing the data-in protocol data unit (PDU) processing cost between Sarwate CRC and Slicing-by-8 (SB8) CRC. Data: Intel Corp.

With the Sarwate CRC generation algorithm, for the smallest I/O size (i.e., 512 B), the protocol cost is about 28% of the total cost, the copy cost is about 14% of the total cost, and the CRC accounts for the remaining 58%. As the PDU size increases, the protocol cost remains the same on a per PDU basis, whereas the CRC and copy costs increase. For the largest I/O size (i.e., 8 KB) the protocol cost is barely 3% of the total cost, whereas CRC is 73% of the total cost and copy accounts for 24%. As you can see, the CRC cost is paid on a per byte basis and increases linearly as the I/O size increases. For a workload of 8 KB, the total CRC cost is about 54,000 cycles, or about 6.5 cycles per byte, while the copy cost is about 2.15 cycles per byte. Thus for the 8 KB PDU size which is a common PDU size, the CRC cost completely dominates the iSCSI protocol processing cost, and is significantly more than the copy cost.

Compare this to the Slicing-by-8 CRC generation algorithm. For the same 8 KB PDU size, each of the data copies and CRC generation represent nearly an equal amount (about 47%) of the total processing cost, while the protocol processing is about 5%. In comparison to the costs of the Sarwate CRC algorithm, the total processing cost for an 8 KB PDU shows a decrease from 73,761 cycles to about 37,579 cycles, resulting in two-times-faster iSCSI processing.

# Conclusion

CRC is currently the biggest bottleneck in iSCSI processing and its impact will increase even further as TCP/IP stacks become more optimized. The Slicing-by-8 algorithm provides an ideal solution to speeding up iSCSI error detection because it reads arbitrarily large amounts of data at a time while optimizing the memory requirement to meet the constraints of specific computer architectures. Our tests show that the Slicing-by-8 software CRC algorithm can deliver immediate benefits by providing three-times-faster performance than the current industry-standard algorithm. The Slicing-by-8 software CRC algorithm has the additional advantage of being able to be implemented in software that runs on commodity processors (as opposed to specialized parallel circuits).

As for the future, Intel expects iSCSI performance to demonstrate near linear scaling with the number of CPU cores available in a system, and that we will see support for data rates greater than what a single 10-Gigabit Ethernet interface will provide. Three factors lead us to this conclusion: (1) the increasing commercial availability of dual-core and multi-core processors; (2) evolving operating system technologies such as receive-side scaling that allow distribution of network processing across multiple CPUs; and (3) the use of multiple iSCSI connections between an initiator and one or more storage targets as supported in many iSCSI implementations today. Thus, the next steps to improving iSCSI performance will involve studying the performance and scalability of iSCSI across multiple CPU cores, as well as the application-level performance of iSCSI storage stacks for both transaction-oriented applications and backup and recovery applications. It's likely the Slicing-by-8 algorithm will become an important solution for helping to increase CRC error detection performance.

### Download the Slicing-by-8 Algorithm Now

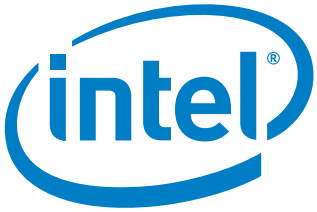
Software developers can download the Slicing-by-8 algorithm now free from Intel at **http://sourceforge.net** to improve the performance of CRC error detection in iSCSI or any other data copying application.

## Learn More

For more information please visit
**http://www.intel.com/technology/comms/perfnet/**
for the following papers:

*A Scalable and High Performance Software iSCSI Implementation* –
USENIX FAST Conference, December, 2005

*A Systematic Approach to Building High Performance Software-based CRC Generators* -
10th IEEE Symposium on Computers and Communications (ISCC'05), June, 2005



www.intel.com


Printed in the United States. 0306/MLG/HBD/PDF 311897-001US